基于静态检测的程序安全漏洞测试

孙　茜[1]　宫云战[2]　杨朝红[2,3]
(1. 北京邮电大学信息工程学院, 北京　100876; 2. 北京邮电大学计算机科学与技术学院, 北京　100876;
3. 装甲兵工程学院信息工程系 北京　100072)

摘　要: 静态分析方法可以自动地提取软件的行为信息,从而检测出软件中的安全漏洞。和其他程序分析方法相比,该方法具有自动化程度高和检测速度快的优点。本文介绍了 Java 语言的安全漏洞的故障模式,说明了类型推断、数据流分析和约束分析等主要静态分析方法及两种特别的分析方法,最后介绍了几种常用的静态代码安全检测工具。

关键词: 静态分析; 安全漏洞; 故障模式; 静态检测工具
中图分类号: TP311.5

引　言

随着社会信息化的不断加深,网络的安全问题[1]越来越突出。安全漏洞是软件中的一种故障模式。此类漏洞的存在可能为他人攻击软件提供方便。而一旦软件被攻击成功,系统就可能发生瘫痪,所造成的危害可能更大,因此,此类漏洞应当尽量避免。

最近几年,静态分析作为一类高效的程序分析方法,受到越来越多的重视。当用户给出语言的抽象语义以后,该类方法能够自动发现满足所有可能(不一定实际存在)执行状态的软件属性。静态分析方法具有自动化程度高,分析速度快,并且可以检验无穷状态系统的优点。尽管静态分析方法可能产生一定的漏报(false negatives)或误报(false positives),但仍然是当今最实用、最有效的安全漏洞检测方法之一。

1　常见的安全漏洞故障模式

程序中的安全漏洞形式多样、种类繁多。如果针对每一个漏洞进行分析、处理,那么安全分析将会有很多重复工作,同时浪费大量系统资源,而且处理速度缓慢。如果透过漏洞的表现形式分析其产生机理,将会发现表面上形态各异的安全漏洞很可能是由同一个原因引起的。NASA 对不同语言存在的安全漏洞做了详细的分析,本文在此基础上对 Java 语言中出现的典型的安全漏洞故障模式进行了分类,主要有如下几类:数值的泄露、垃圾回收器访问权限问题、对象变量合并、方法存储静态域问题和有关域的相关问题。

1.1　数值的泄露

数值的泄露有返回值泄露和 Static 方法中的数组泄露等。

返回值泄露是指返回一个与存储在对象域中的易变对象的相关值时可能泄漏该对象的内部表示。如果实例被不受信任的代码访问,和对易变对象的未受限制的修改将威胁程序安全性或一些重要性质。返回一个对象的复本在很多情况下是一种好的方法。例如,当返回一个对象的引用时可能导致将引用所调用的对象内部信息同时返回,是 private 的信息被对象外的内容调用。

Static 方法中的数组泄露指的是公共静态方法返回与类的静态域中的一个数组相关内容。任何调用这个方法的代码都可以自由的修改这个基本的数组。修补这个漏洞的方法之一是返回对这个数组的副本。

1.2　对象变量合并

代码将与外部的易变对象相关的内容存储到对象内部中。如果实例被不受信任的代码访问,或对对象的不受限制的改变,将威胁安全性和其他一些重要性质。在很多情况下,存储一个对象的复本是

收稿日期: 2007-05-15
基金项目: 国家"863"计划(2006AA01Z184)
第一作者: 女,1982 年生,硕士生
E-mail: sq82child@gmail.com

一种更好的方法。因为可能有其他外部引用可以通过访问外部对象来访问内部信息。

1.3 垃圾回收器访问权限问题

一个类的垃圾回收器应该是 protected 访问的,不应该是 public 访问的。

对于一个类,垃圾回收器应该是保护访问的,如果是 public 的,可能会被攻击,导致不正确的释放空间。同时不建议使用垃圾回收器,如果必须释放时,最好自己编写释放代码,因为无法确定垃圾回收器释放的时间。

1.4 方法存储静态域问题

代码存储异变对象的相关信息到静态域中。如果对异变对象进行不受限制的修改,可能会威胁安全性或者其他的重要性质。存储对象的复本是一种很好的方法。

使用 static 关键字的目的:(1)只想用一个存储区域来保存一个特定的数据;(2)需要一个特殊的方法,它没有与这个类的任何对象关联。

所以将其他对象存储到对象的静态域中的时候,可能导致特定数据被访问,或者为该对象增加了关联,可能产生安全漏洞。

1.5 有关域的相关问题

有关域的相关问题主要有域的非 final 问题、可变数组域、可变 Hash 表、域与接口问题等。

域的非 final 问题是如果一个对象中的域为 final,说明不能改变这个变量的指向,否则该变量域可能被恶意代码所更改,或者由于其他包的一些的异常所修改导致安全漏洞。

Static 方法中的数组泄露指的是公共静态方法返回与类的静态域中的一个数组相关内容。任何调用这个方法的代码都可以自由的修改这个基本的数组。修补这个漏洞的方法之一是返回对这个数组的副本。

可变数组域指的是与数组关联的 final 静态域可能被恶意代码访问,或者收到其他一些包的异常影响。这些代码可以自由地修改数组中的内容。

可变 Hash 表是与 Hash 表关联的 final 静态域可能被恶意代码访问,或者收到其他一些包的异常影响。这些代码可以自由的修改 Hash 表中的内容。

域与接口问题指的是 final 静态域被定义在与可变对象相关的接口,例如:数组或者 Hash 表。该可变对象可以被恶意代码或者其他包中的异常所改变。为了解决这个问题,可以把该域从类中移出,并且进行包保护来避免攻击。

2 静态检测安全漏洞的方法

静态检测使用静态分析技术,直接分析程序的源代码,通过词法分析、语法分析和静态语义分析,检测程序中潜在的安全漏洞。目前,静态分析主要有类型推断、数据流分析和约束分析 3 种方法。

2.1 类型推断

程序语言的类型系统包括一种定义类型的机制,一套有关类型等价、类型相容和类型推理的规则。在将运算符作用于运算对象,执行赋值,或者把实际参数传递给子程序时,都存在着类型是否合适的问题。类型推断是一个处理过程,其目的是保证每个操作都是针对一组数目正确,类型合适的对象进行,以保证操作的有效性。

如果语言能贯彻一套规则,保证任何操作都不会作用到与之不相容的对象上,这种语言就称为是强类型语言。

类型推断可以检查类型错误,选择合适的操作,根据情况确定必要的类型转换。类型推断方法具有简单、高效的特点,非常适合软件安全漏洞的快速检测。已有的采用类型推断方法检测的安全漏洞主要有 C 程序中的格式化字符串漏洞[2]、操作系统内核中的权限检查[3]以及操作系统内核中不安全的指针使用[4]等。

2.2 数据流分析

数据流分析是一项编译时使用的技术,它能从程序代码中收集程序的语义信息并通过代数的方法在编译时确定变量的定义和使用。数据流分析被用于解决编译优化、程序验证、调试、测试、并行、向量化和串行编程环境等问题。数据流分析是通过对变量构造定义—引用对来实现的。

数据流分析在安全检测中有着广泛的用途。应用数据流分析技术,文献[5]检测 C/C++程序中的多种安全漏洞,文献[6]检测 C 程序中的数组越界漏洞,文献[7]对程序安全时态属性进行部分验证。

2.3 约束分析

约束分析方法将程序分析过程分为约束产生和约束求解两个阶段,前者利用约束产生规则建立变量类型或分析状态之间的约束系统,后者对这些约束系统进行求解[8]。

约束系统可以分为等式约束、集合约束和混合

约束三种形式。等式约束的约束项之间只存在相等关系。集合约束把每个程序变量看成一个值集,变量赋值被解释为集合表达式之间的包含关系。混合约束系统由部分等式约束和部分集合约束组成。

约束分析在安全检测中的应用也很广泛。文献[9]利用集合约束方法检测 C 程序中的缓冲区溢出漏洞。

2.4　三种主要静态分析方法的比较

以上介绍的三种主要静态分析方法都是通过解释程序的抽象语义,建立程序属性的数学模型,再通过求解这个数学模型,确定程序的属性。相比较而言,约束分析具有最强的检测能力和最慢的检测速度,适合进行软件的安全检测;数据流分析具有较强和较快快的检测速度,适合检查需要考虑控制流信息而且变量属性之间的操作十分简单的静态分析问题;类型推断则具有最弱的检测能力和最快的检测速度,适合检查属性域有限而且与控制流无关的安全属性。

2.5　其他分析方法

除了以上三种主要的静态分析方法以外,还有美国斯坦福大学的 Ken Ashcraft 和 Dawson Engler 提出的使用写入程序编译扩展发现安全漏洞方法[10]和杨军峰等人提出的 MECA 静态检测安全属性的方法[11]。

2.5.1　用写入程序编译扩展发现安全漏洞(MC)

在这个方法中,程序员写入与编译器相关的系统规范的扩展,来检验他们的代码错误。这个方法发现了 Linux 和 OpenBSD 中超过 100 个安全漏洞,其中 50 多个导致了核心修补。这个方法一个特别的特征是当我们错过应该检测的代码动作时,这个方法会自动进行检测。

MC 发现安全漏洞是通过一个范围检测器和其他两个检测器。范围检测器是当从不被信任的文件中读取的整数在没有被检查的情况下被用于危险的方式的时候发出警告。这个检验器证明了这个方法的有效性。另外两个检验器证明了这个方法的普遍性。其中第一个当从不被信任的文件中读取的端点被反引用时发出警告。第二个当用户引起内核中的非安全性错误,可能对系统产生恶意的攻击时发出警告。

2.5.2　一种用于静态检测安全属性的系统和语言(MECA)

MECA 是一个批注系统,它是一个与灵活的,强功能的批注传播框架耦合的可扩充的批注语言。MECA 由一个发射器、一个检索器、一个批注传播器和一个或更多的检验扩展组成。发射器使用一个修订好版本的 GCC3.1 来分析被检验的系统的源代码和它的抽象语法树的相关批注。抽象语法树是序列化的并且与调用图表和包含所有函数指针分配的文件一起堆放在下一步处理过程的危险上。检索器检索到这些树并用抽象语法树对每个函数建立一个控制流图。CFG 是与全局调用图表相关联的,然后通过传播器进行处理。传播器在整个全局调用图表中传播批注,然后在上面运行已经给定的检验扩展。

MECA 把现有的技术与一些新颖的技术结合起来,具有一个简单且强功能的批注原语,使用静态分析来检测故障批注。

3　静态代码安全检测工具

静态分析技术通过发现源代码中的安全漏洞来防止入侵攻击。静态分析程序时不需要执行所测试的程序,它扫描所测试程序的正文,对程序的数据流和控制流进行分析,然后产生非常人性化的错误报告,会告诉用户发生错误的类型、位置并提出改正的建议,因此可以帮助用户改进软件质量。

目前有许多代码安全检测工具实现了静态分析技术。下面介绍 splint 和 flawfinder 这两款源码检查工具。

PC-Lint 是一种静态代码检测工具,它可以检查出那些虽然完全合乎语法要求,但很可能存在潜在的、不易发现的错误;PC-lint 可以在检查当前文件的同时检查所有与之相关的文件,从而从整个项目的角度来检测问题;PC-lint 支持几乎所有流行的编辑环境和编译器,比如 Borland C++、GCC、VC、VC.net、Source insight 等等,支持 16/32/64 的平台环境;PC-lint 支持 Scott Meyes 的名著(Effective C++/More Effective C++)中说描述的各种提高效率和防止错误的方法。

Flawfinder 是 2001 年发布的基于 Python 语言开发的用来辅助进行安全审查 C/C++ 程序的工具 Ll。它内嵌了一些常见的类似于 ITS4 的程序漏洞数据库,比如缓冲区溢出、格式化串漏洞等,并且扫描速度快,是程序静态检查中极有竞争力的工具。Flawfinder 是遵循 GPL2 协议开发的,运行于类 Unix 平台。Flawfinder 源程序公开,是一款比较好的在程序发布前找出其潜在漏洞的工具。

除了上面介绍的 splint 和 flawfinder 这两款工具外,还有 RATS(一款安全审计工具)以及 ITS4(静态漏洞扫描工具)等工具可用。

4 结束语

静态分析是一种重要的检测安全漏洞的方法。相对于运行时的方法,静态分析没有运行时的开销,能在软件交付使用前检测出安全漏洞。但是,它不是万能的。没有一种检测工具能够消除所有的安全漏洞,静态检测工具也不例外。

参考文献:

[1] 夏一民,罗军,张民选. 基于静态分析的安全漏洞检测技术研究[J]. 计算机科学, 2006, 33: 279 - 282.

[2] SHANKAR U, TALWAR K, FOSTER J S, et al. Detecting format string vulnerabilities with type qualifiers [C]. USENIX Security Symposium, USA, 2001.

[3] ZHANG Xiaolan, EDWARDS A. Using CQUAL for static analysis of authorization hook[C]. USENIX Security Symposium, USA, 2002.

[4] JOHNSON R, WAGNER D. Finding user/kernel pointer bugs with type inference[C]. USENIX Security Symposium, 2004.

[5] LAROCHELLE D. Statically detecting likely buffer overflow vulnerabilities[C]. USENIX Security Symposium, USA, 2001.

[6] XIE Yichen, CHOU A, ENGLER D. ARCHER: Using symbolic pathsensitive analysis to detect memory access errors[C]. ESEC/FSE'03, Helsinki, Finland, September, 2003.

[7] DAS M, LEMER S, SEIGLE M. ESP: path-sensitive program verification in polynomial time[C]. ACM PLDI, Germany, 2002.

[8] AIKEN A. Introduction to set const raint 2 based program analysis[J]. Science of Computer Programming, 1999, 35(2): 79 - 111.

[9] WAGNER D, FOSTER J, BREWER E, et al. A first step towards automated detection of buffer overrun vulnerabilities[C]. Network and Distributed System Security Symposium, USA, 2000.

[10] ASHCRAFT K, ENGLER D. Using programmer-written complier extensions to catch security holes[C]. IEEE Symposium on Security and Privacy, Oakland, California, 2002.

[11] YANG Junfeng, KREMENEK T, XIE Yichen, et al. Dawson Engler MECA: an extensible, expressive system and language for statically checking security properties [C]. In 10th ACM Conf on Computer and Communications Security, 2003.

Software security vulnerability detection approach based on static analysis

SUN Qian[1] GONG YunZhan[2] YANG ZhaoHong[2,3]

(1. College of Information Engineering; Beijing University of Posts and Telecommunications, Beijing 100876;
2. College of Computer Science and Technology, Beijing University of Posts and Telecommunications, Beijing 100876;
3. Department of Information Engineering, Academy of Armored Force Engineering, Beijing 100072, China)

Abstract: Static analysis can find security vulnerabilities by automatically deriving information about the behavior of software. Comparing with other program analysis methods, static analysis method can detect security vulnerabilities automatically and effectively. This paper introduces the fault pattern of security vulnerability in Java language, and then presents the main static analysis methods and other two special methods. Lastly, some popular tools for detection of security vulnerability are listed.

Key words: static analysis; security vulnerability; fault pattern; static detecting tool